Advanced Computer Systems Project 2 Report

Ivan Cheng (ECSE)

Rensselaer Polytechnic Institute

ECSE 4320 Advanced Computer Systems

Dr. Tong. Zhang

October 1, 2025

Setup/methodology

The experiments were performed on an Intel Core i7-11850H laptop.
OS: Ubuntu on WSL2
Compiler: g++ with -03 -std=c++17
SMT enabled, but experiments are single threaded

Experimental Knobs:
Patterns: sequential vs. random
Strides: 64B, 256B, 1024B
R/W ratios: 100R, 100W, 70/30, 50/50
Intensity: 1-8 threads
Page size: 4KB, huge pages are 2MB

Measurement:
Latency from pointer-chasing (ns + cycles per access)
Bandwidth in GB/s
Experiments repeated >= 3 times, averages reported

Results & Analysis

Zero-queue baselines

bytes	pattern	strideB	ns_per_access	cycles_per_access
4096	chase	64	4.10156	10.2539
8192	chase	64	4.03442	10.0861
16384	chase	64	4.39453	10.9863
32768	chase	64	4.15649	10.3912
57344	chase	64	5.73556	14.3389
110592	chase	64	5.7147	14.2867
212992	chase	64	5.68824	14.2206
409600	chase	64	5.83875	14.5969

786432	chase	64	5.68345	14.2086
1511424	chase	64	6.14649	15.3662
2912256	chase	64	5.99178	14.9795
5619712	chase	64	5.9945	14.9862
10829824	chase	64	6.11753	15.2938
20885504	chase	64	6.9635	17.4087
40271872	chase	64	8.54962	21.3741
77656064	chase	64	8.77782	21.9446
1.5E+08	chase	64	8.9482	22.3705
2.89E+08	chase	64	9.10721	22.768
5.57E+08	chase	64	8.96056	22.4014
1.07E+09	chase	64	8.95298	22.3825

Based on the data, the range is 4ns-9ns per access.
This is expected as the smaller working sets that fit in L1 are ~4ns.
Larger sets that spill into L2/L3 are ~5-7 ns.
DRAM are ~8-9ns.
This matches hierarchical behavior as latency in ns/access increases as working set grows.

Pattern & granularity sweep

bytes	pattern	strideB	GBps	ns_per_elem
65536	random	64	0.771213	15.5599

196608	random	64	3.00216	3.99712
569344	random	64	10.9888	1.09202
1667072	random	64	12.5344	0.957367
4894720	random	64	11.7649	1.01998
14385152	random	64	14.2327	0.843129
42278912	random	64	13.0147	0.922033
1.24E+08	random	64	13.087	0.916943
3.65E+08	random	64	13.2944	0.902633
1.07E+09	random	64	12.3263	0.97353

bytes	pattern	strideB	GBps	ns_per_elem
65536	random	256	0.637303	18.8293
196608	random	256	2.30041	5.21647
569344	random	256	7.21805	1.6625
1667072	random	256	13.6199	0.881066
4894720	random	256	20.7756	0.577602
14385152	random	256	22.1875	0.540845
42278912	random	256	23.8921	0.502258
1.24E+08	random	256	23.9641	0.500748

3.65E+08	random	256	23.5472	0.509615
1.07E+09	random	256	20.484	0.585824

bytes	pattern	strideB	GBps	ns_per_elem
65536	random	1024	0.799761	15.0045
196608	random	1024	2.18588	5.48977
569344	random	1024	5.70994	2.1016
1667072	random	1024	16.7789	0.715186
4894720	random	1024	40.9448	0.293078
14385152	random	1024	60.4955	0.198362
42278912	random	1024	79.8904	0.150206
1.24E+08	random	1024	89.7967	0.133635
3.65E+08	random	1024	83.411	0.143866
1.07E+09	random	1024	75.6237	0.15868

bytes	pattern	strideB	GBps	ns_per_elem
65536	seq	64	0.645746	18.5832

196608	seq	64	2.2521	5.32837
569344	seq	64	5.23615	2.29176
1667072	seq	64	11.6452	1.03047
4894720	seq	64	14.1643	0.847199
14385152	seq	64	13.0277	0.921116
42278912	seq	64	12.9246	0.928463
1.24E+08	seq	64	13.222	0.907575
3.65E+08	seq	64	13.3334	0.899993
1.07E+09	seq	64	12.1813	0.985116

bytes	pattern	strideB	GBps	ns_per_elem
65536	seq	256	0.606565	19.7835
196608	seq	256	2.31183	5.1907
569344	seq	256	7.62628	1.57351
1667072	seq	256	15.5769	0.770373
4894720	seq	256	20.119	0.596452
14385152	seq	256	22.2857	0.538463
42278912	seq	256	23.1877	0.517516
1.24E+08	seq	256	23.9606	0.500822

3.65E+08	seq	256	23.7591	0.50507
1.07E+09	seq	256	20.7368	0.578681

bytes	pattern	strideB	GBps	ns_per_elem
65536	seq	1024	0.759495	15.8
196608	seq	1024	2.69244	4.45692
569344	seq	1024	6.57441	1.82526
1667072	seq	1024	16.0398	0.748138
4894720	seq	1024	36.6554	0.327373
14385152	seq	1024	57.8052	0.207594
42278912	seq	1024	79.7598	0.150452
1.24E+08	seq	1024	83.0983	0.144407
3.65E+08	seq	1024	84.4556	0.142087
1.07E+09	seq	1024	59.402	0.202013

In my data, we see that in the 64B dataset the bandwidth rises from 0.85GB/s to 13GB/s at large sizes with ns/element falling close to 0.9ns. This case shows the best efficiency baseline. At the 256B dataset the bandwidth goes up to 24GB/s at peak and latency goes further down to 0.5 ns/element. This shows that there are fewer memory requests per byte of allocated space. At 1024B dataset bandwidth peaks at 90GB/s with ns/element at 0.13ns. Demand is massively reduced and GBps is inflated as a lot of data is skipped. Large strides reduce cache pressure and memory seems to be fast.
Between the random and sequential access datasets we can see that the sequential access at small sizes GBps is modest but increase quickly. As working set DRAM bandwidth is fully utilized, and ns/elements drops steadily until bandwidth saturates.

In random access dataset GBps is significantly lower than sequential at the same sizes. In addition, ns/elements is higher which shows stalls. Growth is slower as size increases. This matches expectations as sequential curves climb higher and flatten out while random curves plateau at a lower ceiling.

Read/Write mix sweep

bytes	rw	strideB	GBps	ns_per_elem
5.37E+08	50R50W	64	1.99261	20.0742
5.37E+08	70R30W	64	1.49436	26.7674
5.37E+08	100R	64	1.06547	37.5421
5.37E+08	100W	64	3.08045	12.9851

In this experiment, the read/write ratio was varied across 100%R, 100%W, 70R/30W, 50R/50W while keeping the stride and size constant. The results show how the system responds to read and write dominated workloads.
In 100%R, throughput was the highest as reads benefit from hardware prefetching and can be pipelined efficiently. Latency per element was correspondingly the lowest, indicating that the memory hierarchy is optimized for streaming read access.
In 100%W performance was lowest. Writes require cache line invalidations and store buffer handling and can’t be prefetched in the same way as reads. As a result, throughput dropped significantly and per element latency increased.
In mixed ratios, writes degraded performance compared to only reads. Overall, this is all expected as the results align with hardware expectations as it is expected that writes degrade performance.

Intensity sweep

bytes	strideB	threads	GBps	ns_per_elem

1.07E+09	64 (T1)	1	13.319	0.900969
1.07E+09	64 (T2)	1	13.2801	0.903605
1.07E+09	64 (T2)	2	23.3406	0.514126
1.07E+09	64 (T4)	1	13.4487	0.892278
1.07E+09	64 (T4)	2	23.3375	0.514193
1.07E+09	64 (T4)	4	28.279	0.424343
1.07E+09	64 (T8)	1	13.3369	0.899757
1.07E+09	64 (T8)	2	23.9071	0.501944
1.07E+09	64 (T8)	4	28.6569	0.418747
1.07E+09	64 (T8)	8	24.3062	0.493701

Based on my results, GB/s raised from T = 1 -> T = 4
As T = 8 it started to flatten out and there was small improvement
ns/element decreased at first and went back up at T = 8.
This is expected as the memory system can keep up to 4 threads of requests efficiently.
At T = 8, DRAM queues are saturated and more concurrency increases queuing,

Working-set size sweep

bytes	stride B	ns_per_access	cycles_per_access

4096	64	4.08936	10.2234
8192	64	4.03442	10.0861
16384	64	4.02527	10.0632
32768	64	4.02374	10.0594
57344	64	5.6536	14.134
110592	64	5.67898	14.1975
212992	64	5.67674	14.1918
409600	64	5.66882	14.1721
786432	64	5.66241	14.156
1511424	64	6.368	15.92
2912256	64	5.97188	14.9297
5619712	64	6.01906	15.0477
1082982 4	64	6.05957	15.1489
2088550 4	64	6.96222	17.4055
4027187 2	64	8.54217	21.3554
7765606 4	64	8.80131	22.0033
1.5E+08	64	8.8509	22.1272
2.89E+0 8	64	8.95922	22.398
5.57E+0 8	64	8.94094	22.3524
1.07E+0 9	64	8.92155	22.3039

Based on my data, my data properly shows the cache locality transitions.
In the L1 cache region, my data is around 4KB to 32KB with ~4ns per access.
This is the expected L1 cache latency.
In the L2 cache region, my data is around 57KB to 512KB with latency rising to 5.6ns per access. This is the L2 hit latency and is larger than L1 but faster than DRAM.
In the L3 cache region, latency jumps to ~6.3ns. From 2MB to 20MB latency is around 15-17 cycles.
In the DRAM region, which is 40 MB and beyond, latency rises to ~8.5-9ns (21-22 cycles).
My results matches how expected cycling and ns per access jumps discretely when going from different levels of cache memory.

Cache-miss impact

bytes	pattern	strideB	elapsed_s
65536	seq	64	2.80E-06
159744	seq	64	7.07E-06
385024	seq	64	3.45E-05
925696	seq	64	0.00012
2236416	seq	64	0.00032
5398528	seq	64	0.000915
13041664	seq	64	0.002804
31506432	seq	64	0.006806
76124160	seq	64	0.017435
1.84E+08	seq	64	0.040932
4.44E+08	seq	64	0.099455

1.07E+09	seq	64	0.236811

bytes	pattern	strideB	elapsed_s
65536	seq	4096	3.67E-07
159744	seq	4096	4.67E-07
385024	seq	4096	1.37E-06
925696	seq	4096	4.30E-06
2236416	seq	4096	1.40E-05
5398528	seq	4096	5.69E-05
13041664	seq	4096	0.000147
31506432	seq	4096	0.00035
76124160	seq	4096	0.000793
1.84E+08	seq	4096	0.001962
4.44E+08	seq	4096	0.005369
1.07E+09	seq	4096	0.013014

bytes	pattern	strideB	elapsed_s
65536	random	64	3.20E-06
159744	random	64	8.43E-06

385024	random	64	4.93E-05
925696	random	64	0.000163
2236416	random	64	0.000519
5398528	random	64	0.002121
13041664	random	64	0.007572
31506432	random	64	0.023785
76124160	random	64	0.073893
1.84E+08	random	64	0.227151
4.44E+08	random	64	0.625386
1.07E+09	random	64	1.6695

bytes	pattern	strideB	elapsed_s
65536	random	4096	4.00E-07
159744	random	4096	4.00E-07
385024	random	4096	9.33E-07
925696	random	4096	2.57E-06
2236416	random	4096	1.64E-05
5398528	random	4096	5.18E-05
13041664	random	4096	0.000158

3150643 2	random	4096	0.000431
7612416 0	random	4096	0.001296
1.84E+0 8	random	4096	0.004
4.44E+0 8	random	4096	0.011638
1.07E+0 9	random	4096	0.032187

Based on my data, latency is lowest in my sequential data access in 64B as as the hardware prefetcher brings in data smoothly. Miss rate is minimized until cache capacity is exceeded.
In my sequential 4KB data, entire cache lines are skipped between accesses. This results in more cache misses as the prefetcher only works in between ~1-2KB regions.
In my random cache data accesses the prefetcher can't predict random jumps so caches for the random 64B is worse compared to the sequential 64B but still better than the 4KB stride. The random 4KB accesses are random and sparse. This inflates the runtime as there are more misses.

TLB-miss impact

bytes	huge	page_stride B	random	elapsed_ s	huge_grante d
4194304	0	64	1	0.00198 9	0
4194304	1	2097152	1	1.87E-06	1
1022771 2	0	64	1	0.00522 6	0
1022771 2	1	2097152	1	5.37E-06	1
2493235 2	0	64	1	0.01566 1	0

24932352	1	2097152	1	2.13E-05	1
60784640	0	64	1	0.042048	0
60784640	1	2097152	1	4.97E-05	1
1.48E+08	0	64	1	0.117143	0
1.48E+08	1	2097152	1	0.000118	1
3.61E+08	0	64	1	0.345566	0
3.61E+08	1	2097152	1	0.00031	1
8.81E+08	0	64	1	0.970923	0
8.81E+08	1	2097152	1	0.00073	1
2.15E+09	0	64	1	2.59477	0
2.15E+09	1	2097152	1	0.002002	1

From my data experiment, we can see that normal pages show superlinear growth over .002s at 4MB to ~2.6s at 2GB. This is what is expected as TLB misses start to dominate as the dataset grows. Huge pages remain stable, staying in the micro/millisecond range at 2GB. The speedup grows with dataset size. This matches theory as with a TLB reach with 4KB pages access past around ~6MB start risking misses.

Latency Percentiles

Workload	QD	P50(ms)	p95(ms)	p99(ms)	p99.9(ms)

4k rand read	8	.029	.051	.057	.108
4k rand read	64	.030	.052	.060	.135

The table shows the percentile latencies for a 4KiB random read workload at QD =8 and QD = 64. The difference in the latencies although small shows the effect of queuing while throughput scales with QD, tail latency grows. For SLA, systems often set performance guarantees at p99/p99.9. This means that if average latency remains below the SLA threshold, tail violations can cause requests to miss the deadline.
Data was collected through DiskSpd with powershell lines
.\diskspd.exe -c1G -b4K -r -d30 -t1 -o8 -L -w0 -Rtext -Sh testfile.dat > results_qd8.txt
.\diskspd.exe -c1G -b4K -r -d30 -t1 -o64 -L -w0 -Rtext -Sh testfile.dat > results_qd64.txt

Conclusion

This project successfully shows cache and memory performance across the hierarchy of a modern CPU. Each experiment produces results that generally show results that are consistent with theory. Despite a few anomalies that occurred such as small working set sizes sometimes showing unstable bandwidth and latency numbers. The results provide a general clear idea on how caching, bandwidth, and address translations impact the performance of a CPU.

Reference

[1] OpenAI, “ChatGPT,” ChatGPT, 2025. https://chatgpt.com/
[2]microsoft, “Releases · microsoft/diskspd,” GitHub, Jun. 13, 2024. https://github.com/microsoft/diskspd/releases